R-trees

R-TREES: A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING By Antonin Guttman

R-tree (A. Guttman SIGMOD’1984)
Spatial structure

**Outline**

- Introduction
- R-Tree Index Structure
- Searching and Updating
  - Searching
  - Insertion
  - Deletion
  - Updates and Other Operations
  - Node Splitting
- Performance Tests
Spatial structure

Introduction

• **Spatial Data Objects**
  – Covers areas in multi-dimensional spaces
  – Traditional indexing methods
    • One-dimensional
    • Represented by point locations
    • Not well for spatial data: map objects like countries
  – Common operation: search for all objects in an area
    • Find all countries that have land within 20 miles of a particular point.
  – Applications:
    • Computer Aided Design (CAD)
    • Geo-data Applications

• **Objective**: a dynamic index structure for objects of non-zero size located in multi-dimensional spaces

• Search structures such as hash tables can not be used as range search is required.

• Structures such as one-dimensional ordering of key values, such as B-trees and ISAM indexes do not work because search space is multi-dimensional.

• **R-Trees**: represent spatial objects by intervals in several dimensions
Spatial structure

Index Structures

• Height-balanced tree
• Dynamic index
  – No periodic reorganization
• A spatial database consists of a collection of tuples representing spatial objects.
• Each tuple has a unique identifier
• Leaf Nodes:
  – Index records
    • \((I, \text{tuple_identifier})\)
      – \(I\) : n-dimensional bounding box of spatial object \(I=(I_0, I_1, ..., I_{n-1})\)
      – \(I_i\) : closed bounded interval \([a, b]\) describing extent of object along dimension \(i\)
  – Pointers to data objects
• Nodes: disk pages
  – \((I, \text{child_pointer})\)
    • Child_pointer : address of a lower node
    • \(I\) : covers all rectangles in the lower nodes
• Parameters:
  – \(M\) : maximum number of entries in one node
  – \(m \leq M / 2\) : minimum number of entries in one node
Spatial structure

Index Structures …

• R-Tree Properties
  – Every leaf node contains between \( m \) and \( M \) index records unless it is the root
  – For each index record \((I, \text{tuple-identifier})\), in a leaf node, \( I \) is the smallest rectangle that spatially contains the \( n \)-dimensional data object represented by the indicated tuple
  – Every non-leaf node contains between \( m \) and \( M \) index records unless it is the root
  – For each entry \((I, \text{child-pointer})\), in a non-leaf node, \( I \) is the smallest rectangle that spatially contains the rectangles in the child node
  – The root node has at least two children unless it is a leaf
  – All leaves appear on the same level.
Spatial structure

Index Structures ...
Spatial structure

Index Structures ...

• M: maximum number of entries that will fit in one node
• m <= M/2 be a parameter specifying the minimum number of entries in a node also called branching factor.
• If N is the number of index records, the Maximum Height of R-Tree

\[ -\left|\log_m N\right| - 1 \]

• Maximum Nodes in R-Tree

\[ -\left|N/m\right| + \left|N/m^2\right| + \ldots + 1 \]

• Worst-case space utilization except the root is: \(m/M\)

• If m increases:
  – Tree height decrease
  – Space utilization improved
  – Tree is wide and space used for leaf nodes (indexes)
  – Can be adjusted during performance tuning
Spatial structure

R-trees: Searching

• The search algorithms descend the tree from the root in a manner similar to a B-Tree.
• The algorithms maintain a tree in a form that eliminate irrelevant regions of the indexed space. And examine only data near by the search area.
• Algorithm Search: Find all index records whose rectangles overlap a search rectangle S, in R-Tree with root node T
  1. [Search subtrees]: If T is not leaf, check each entry E whether EI overlaps S. For all overlapping entries, Search tree whose root node is pointed by Ep
  2. [Search leaf node]: If T is leaf, check all entries E whether EI overlaps S. If so return E.
• (rectangle part of index entry E is EI, tuple-identifier or child-pointer is Ep)
Spatial structure

R-trees : Insertion

• Similar to the insertion in B-Tree.
  – Nodes that overflow are split, splits propagate up the tree.

• Algorithm **Insert** : Insert a new index entry E into an R-Tree

  1. **[Find position]** : Invoke **ChooseLeaf** to select a leaf node L to place E
  2. **[Add record to leaf node]** : If L has room install E. Otherwise invoke **SplitNode** to obtain L and LL
  3. **[Propagate changes upward]** : Invoke **AdjustTree** on L and LL is split performed
  4. **[Grow tree taller]** : If after propagation root node splited, create a new root whose chlids are resulting nodes
**Spatial structure**

**R-trees : Insertion**

- Algorithm **ChooseLeaf**: Select leaf node to place new entry \( E \)

1. **[Initialize]**: Set \( N \) to be root node
2. **[Leaf Check]**: If \( N \) is leaf, return \( N \)
3. **[Choose Subtree]**: Choose \( F \) in node \( N \) where \( F_I \) needs least enlargement to include \( E_I \)
4. **[Descend until a leaf is reached]**: Set \( N \) be the child node pointed by \( F_p \) and go step 2
Algorithm ChooseSubtree

CS1 [Initialize] Set N to be the root node

CS2 [Leaf check]
If N is a leaf,
    return N
else
    [Choose subtree]
    Choose the entry in N whose rectangle needs least area enlargement to include the new data. Resolve ties by choosing the entry with the rectangle of smallest area
end

CS3 [Descend until a leaf is reached]
Set N to be the child node pointed to by the child pointer of the chosen entry. Repeat from CS2
Spatial structure

ChooseSubtree
Spatial structure
Spatial structure

ChooseSubtree

Root
Since $N = \text{Leaf}$,
stop and return $N$
Spatial structure

Insert
Spatial structure

**Insertion: AdjustTree**

- **Algorithm AdjustTree**: Ascend from leaf node L to root, adjust covering rectangles and propagate node splits if necessary

1. **[Initialize]**: Set N=L, if L was split before set NN resulting second node
2. **[Check if done]**: If N is root stop
3. **[Adjust covering rectangle in parent]**: Let P be parent node of N, E^N be N’s entry in P. Adjust E^N.I to enclose all entry rectangles in N
4. **[Propagate node split upward]**: Create E^NN with E^NN.p pointing to NN and E^NN.I encloses rectangles in NN, add E^NN to P if there is room else invoke **SplitNode** to produce P and PP
5. **[Move up to next level]**: Set N=P and NN=PP, go to step 2
Delete

- Algorithm **Delete**: Remove index record E from R-Tree
  1. [Find node]: Invoke **FindLeaf** to find node L that contains E. Stop if not found
  2. [Delete Record]: Remove E from L
  3. [Propagate changes]: Invoke **CondenseTree**, pass L
  4. [Shorten tree]: If root has only one child after adjusting, make child node new root
Spatial structure

**Delete : FindLeaf**

- Algorithm **FindLeaf** : Find node containing index entry E in R-Tree with root node T

1. **[Search Subtrees]** : If T is no leaf, check each entry F in T if FI overlaps EI. For each such entry invoke **FindLeaf** on tree with root pointed by Fp until E is found or all entries checked.
2. **[Search leaf node for record]** : If T is leaf, check each entry if it matches E. If E found return T.
Spatial structure

Deletion: CondenseTree

•Algorithm **CondenseTree**: Eliminate leaf node L from which an entry has been deleted, if it has too few entries and reallocate them. Propagate upward. Adjust covering rectangles on the path to make them smaller

1. **[Initialize]**: Set N=L, Set Q, set of eliminated nodes, empty
2. **[Find parent]**: If N is root go to step 6, else let P parent of N and E_N is entry in P
3. **[Eliminate under-full node]**: If N has fewer than m entries, delete E_N from P and add N to Q
4. **[Adjust covering rectangles]**: If N not eliminated, adjust E_N to cover all entries in N
5. **[Move up one level]**: Set N=P, go to step 2
6. **[Re-insert orphaned entries]**: Re-insert all entries in Q. Use algorithm **Insert** but higher level eliminated nodes must be placed higher in tree.
Spatial structure

Updates and other operations

• If a data tuple updated:
  – Delete its index record
  – Update its covering rectangle
  – Re-insert

• Area Selection
• Range Deletion
Spatial structure

Node splitting

• Add new entry to full node with M entries by dividing M+1 entries between two nodes
• Criteria: total area of resulting rectangles after split minimize as in ChooseLeaf

Bad split

Good split
Spatial structure

Node splitting Algorithm

• Exhaustive Algorithm
  – Generate all possible groupings and choose the best
  – \(2^M-1\) possibilities
  – Too slow for large \(M\) (\(M=50*\) reasonable)
  – find the area with global minimum
  – CPU cost too high

• A Quadratic-Cost Algorithm
  – Quadratic cost in \(M\)

• A Linear-Cost Algorithm
  – Linear cost in \(M\)
  – Same as Quadratic Split but
    • Choose any the remaining entries for first and third steps

*每个矩形由4个字节的4个参数组成，设每个指针也需要4个字节，共20个字节，20*50=1000字节，一页有1024个字节
Spatial structure

Node splitting: A Quadratic-Cost Algorithm

• Finds smallest-area split, but not guaranteed
  – Quadratic cost in M.
  – Pick two of M+1 nodes that wastes most area if both were put in same group.
  – Assign remaining entries one at a time
    • Calculate required area expansion
    • Add to group with least enlargement required
    • Pick next entry with greatest difference between groups
Spatial structure

Algorithm QuadraticSplit

[Divide a set of $M+1$ index entries into two groups]

QS1  [Pick first entry for each group ]

Invoke PickSeeds to choose two entries, each be first entry of each group.

QS2  [Check if done]

Repeat

DistributeEntry

until all entries are distributed or one of the two groups has $M-m+1$ entries (so that the other group has $m$ entries)

QS3  [Select entry to assign ]

If entries remain, assign them to the other group so that it has the minimum number $m$ required.
Algorithm PickSeeds

[Choose two entries to be the first entries of the groups]

PS1  [Calculate inefficiency of grouping entries together]

For each pair of entries $E_1$ and $E_2$, compose a rectangle $R$ including $E_1$ rectangle and $E_2$ rectangle

Calculate $d = \text{area}(R) - \text{area}(E_1 \text{ rectangle}) - \text{area}(E_2 \text{ rectangle})$

PS2  [Choose the most wasteful pair ]

Choose the pair with the largest $d$

[the seeds will tend to be small, if the rectangles are of very different size (and) or the overlap between them is high]
Algorithm DistributeEntry

[Assign the remaining entries by the criterion of minimum area]

DE1 Invoke PickNext to choose the next entry to be assigned

DE2 Add it to the group whose covering rectangle will have to be enlarged least to accommodate it. Resolve ties by adding the entry to the group with the smallest area, then to the one with the fewer entries, then to either

Algorithm PickNext

[chooses the entry with best area-goodness-value in every situation]

DE1 For each entry E not yet in a group, calculate $d_1 =$ the area increase required in the covering rectangle of Group 1 to include E Rectangle. Calculate $d_2$ analogously for Group 2

DE2 Choose the entry with the maximum difference between $d_1$ and $d_2$
Spatial structure

ChooseSubtree

Diagram showing a spatial structure with nodes labeled A to M and a subtree highlighted.
Spatial structure
Spatial structure

ChooseSubtree

N = Leaf
Node is full
Spatial structure

ChooseSubtree

QuadraticSplit
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds

<table>
<thead>
<tr>
<th>E1</th>
<th>E2</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds

<table>
<thead>
<tr>
<th>E1</th>
<th>E2</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>K</td>
<td>1.0</td>
</tr>
<tr>
<td>J</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds

<table>
<thead>
<tr>
<th>E1</th>
<th>E2</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>K</td>
<td>1.0</td>
</tr>
<tr>
<td>J</td>
<td>L</td>
<td>1.1</td>
</tr>
<tr>
<td>J</td>
<td>M</td>
<td>4.0</td>
</tr>
<tr>
<td>J</td>
<td>Y</td>
<td>1.7</td>
</tr>
<tr>
<td>K</td>
<td>L</td>
<td>0.1</td>
</tr>
<tr>
<td>K</td>
<td>M</td>
<td>3.6</td>
</tr>
<tr>
<td>K</td>
<td>Y</td>
<td>3.5</td>
</tr>
<tr>
<td>L</td>
<td>M</td>
<td>5.6</td>
</tr>
<tr>
<td>L</td>
<td>Y</td>
<td>4.7</td>
</tr>
<tr>
<td>M</td>
<td>Y</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E1</th>
<th>E2</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>K</td>
<td>1.0</td>
</tr>
<tr>
<td>J</td>
<td>L</td>
<td>1.1</td>
</tr>
<tr>
<td>J</td>
<td>M</td>
<td>4.0</td>
</tr>
<tr>
<td>J</td>
<td>Y</td>
<td>1.7</td>
</tr>
<tr>
<td>K</td>
<td>L</td>
<td>0.1</td>
</tr>
<tr>
<td>K</td>
<td>M</td>
<td>3.6</td>
</tr>
<tr>
<td>K</td>
<td>Y</td>
<td>3.5</td>
</tr>
<tr>
<td>L</td>
<td>Y</td>
<td>4.7</td>
</tr>
<tr>
<td>M</td>
<td>Y</td>
<td>0.9</td>
</tr>
</tbody>
</table>
ChooseSubtree
QuadraticSplit
PickSeeds
DistributeEntry
PickNext

<table>
<thead>
<tr>
<th>Entry</th>
<th>d1</th>
<th>d2</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Spatial structure**
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds
DistributeEntry
PickNext

<table>
<thead>
<tr>
<th>Entry</th>
<th>d1</th>
<th>d2</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>K</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>M</td>
</tr>
</tbody>
</table>
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds
DistributeEntry
PickNext

G1  G2
L  M
Y

Entry  d1  d2
J  1.0  2.0
K  0.5  2.1
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds
DistributeEntry
PickNext

Entry d1 d2
J 1.0 1.2
K 0.5 2.3
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds
DistributeEntry
PickNext

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>K</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry</th>
<th>d1</th>
<th>d2</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>1.0</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds
DistributeEntry
PickNext

G1 | G2
---|---
L  | M
K  | Y

Entry | d1 | d2
------|----|----
J     | 0.7| 1.2
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds
DistributeEntry
PickNext

<table>
<thead>
<tr>
<th>G1</th>
<th>G2</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>K</td>
<td>Y</td>
</tr>
<tr>
<td>J</td>
<td></td>
</tr>
</tbody>
</table>
Spatial structure

ChooseSubtree
QuadraticSplit
PickSeeds
DistributeEntry
PickNext

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>G2</td>
</tr>
<tr>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>K</td>
<td>Y</td>
</tr>
<tr>
<td>J</td>
<td></td>
</tr>
</tbody>
</table>
Spatial structure

Node splitting …

• **A Linear-Cost Algorithm**
  – This algorithm is linear in M and in the number of dimensions.
  – Linear split is identical to quadratic split but uses a different version of **PickSeeds**. **PickNext** simply chooses any of the remaining entries.

• **Algorithm LinearPickSeeds**
  – LPS1. [Find extreme rectangles along all dimensions] Along each dimension find an entry whose rectangle has the highest low side, and the one with the lowest high side. Record the separation.
  – LPS2. [Adjust for shape of rectangle cluster] Normalize the separations, by dividing by the width of the entire set along the corresponding dimension.
  – LPS3. [Select the most extreme pair] Choose the pair with the greatest normalized separation along any dimension.
PERFORMANCE TESTS

• Test Conditions:
  – Implement R-Tree in C under Unix
  – Data set: VLSI layout data contains 1027 rectangles.
  – Different values for M and m to test node splitting algorithms
    • Tested m values: M/2, M/3, and 2
    • Two-dimensional data
    • Data: VLSI layout data of CENTRAL circuit cell with 1057 rectangles
      – Insert Test: last 10% of data
      – Select Test: 100 searches each retrieves 5% of data
      – Delete Test: 10% of indexed records

<table>
<thead>
<tr>
<th>Bytes per Page</th>
<th>Max Entries per Page (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>6</td>
</tr>
<tr>
<td>256</td>
<td>12</td>
</tr>
<tr>
<td>512</td>
<td>25</td>
</tr>
<tr>
<td>1024</td>
<td>50</td>
</tr>
<tr>
<td>2048</td>
<td>102</td>
</tr>
</tbody>
</table>
Spatial structure

**PERFORMANCE TESTS**

**CPU cost of inserting records**

- \( E \) = Exhaustive algorithm
- \( Q \) = Quadratic algorithm
- \( L \) = Linear algorithm

**Search performance**

- Pages touched per qualifying record
- \( E \) = Exhaustive algorithm
- \( Q \) = Quadratic algorithm
- \( L \) = Linear algorithm

<table>
<thead>
<tr>
<th>Bytes per page</th>
<th>CPU msec per insert</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>5</td>
</tr>
<tr>
<td>256</td>
<td>10</td>
</tr>
<tr>
<td>512</td>
<td>20</td>
</tr>
<tr>
<td>1024</td>
<td>50</td>
</tr>
<tr>
<td>2048</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes per page</th>
<th>Pages touched</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>6</td>
</tr>
<tr>
<td>256</td>
<td>4</td>
</tr>
<tr>
<td>512</td>
<td>2</td>
</tr>
<tr>
<td>1024</td>
<td>1</td>
</tr>
<tr>
<td>2048</td>
<td>0.5</td>
</tr>
</tbody>
</table>
Spatial structure

PERFORMANCE TESTS

CPU cost of deleting records

Search performance  CPU cost

Graphs showing CPU cost and search performance for different algorithms.
Conclusions

• The R-Tree structure is shown to be useful for indexing spatial data objects that have non-zero size.
• Nodes corresponds to disk pages of reasonable size.